FormCalc and OPP

Thomas Hahn

Max-Planck-Institut für Physik
München

The FeynSystem

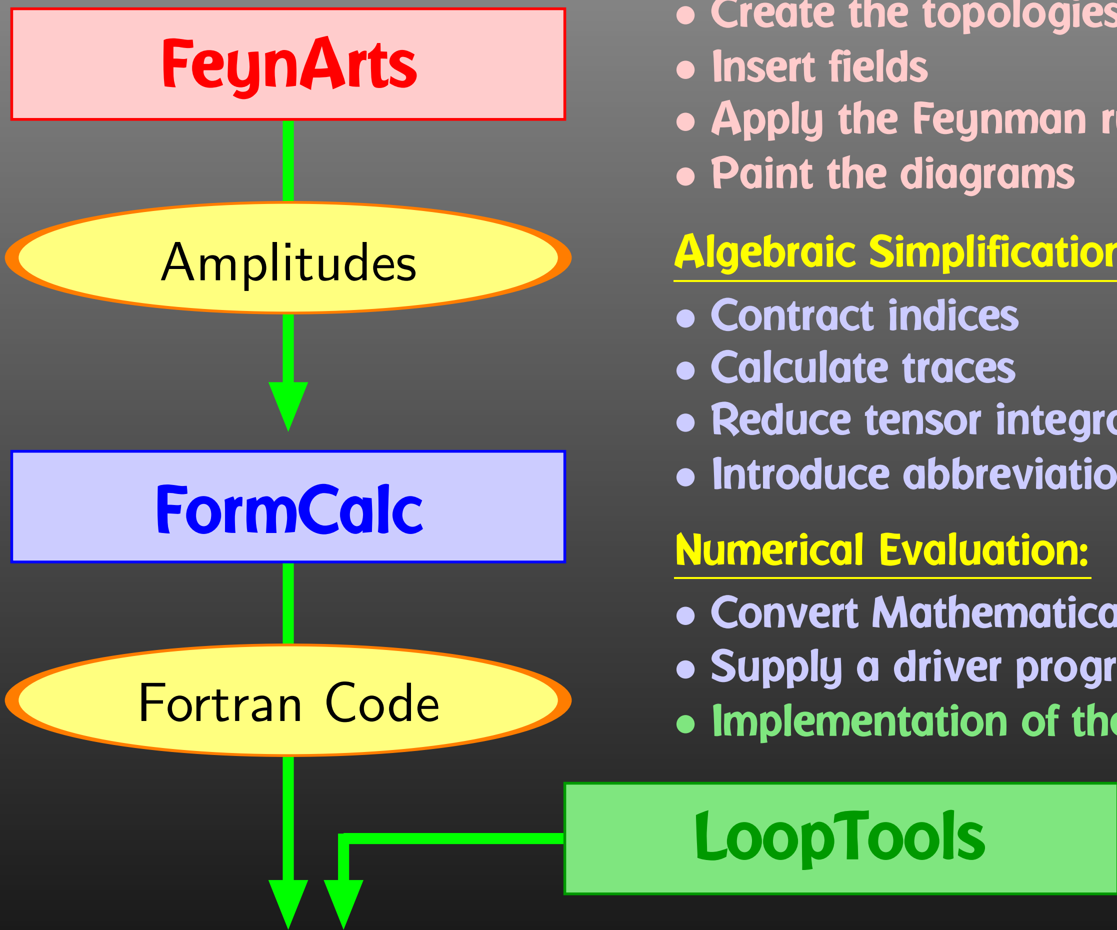


Diagram Generation:

- Create the topologies
- Insert fields
- Apply the Feynman rules
- Paint the diagrams

Algebraic Simplification:

- Contract indices
- Calculate traces
- Reduce tensor integrals
- Introduce abbreviations

Numerical Evaluation:

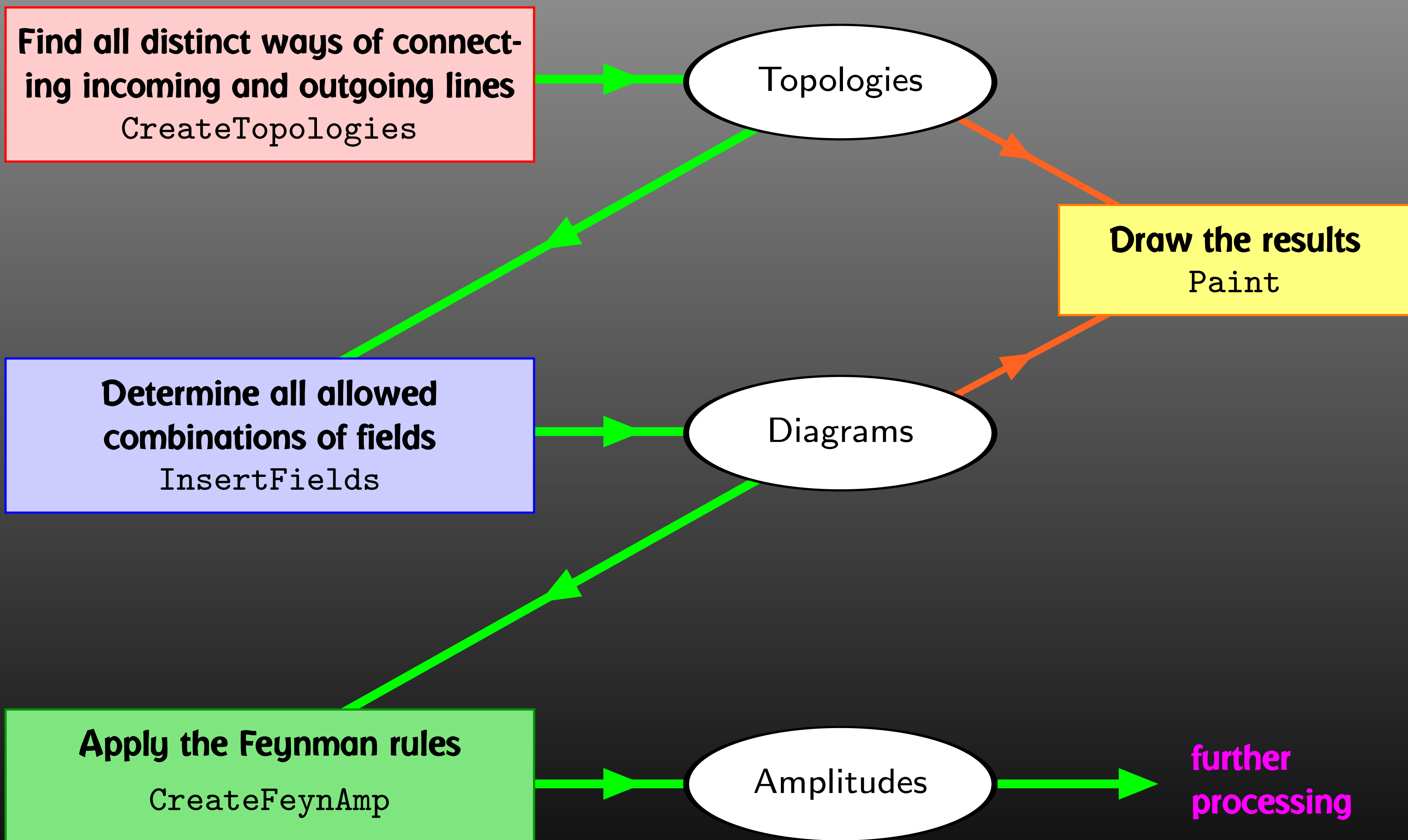
- Convert Mathematica output to Fortran code
- Supply a driver program
- Implementation of the integrals

Symbolic manipulation (Computer Algebra) for the structural and algebraic operations.

Compiled high-level language (Fortran) for the numerical evaluation.

$|\mathcal{M}|^2 \longrightarrow$ **Cross-sections, Decay rates, . . .**

FeynArts


Find all distinct ways of connecting incoming and outgoing lines
CreateTopologies
Topologies
Draw the results
Paint
Determine all allowed combinations of fields
InsertFields
Diagrams
Apply the Feynman rules
CreateFeynAmp
Amplitudes
further processing

Three Levels of Fields

Generic level, e.g. `F, F, S`

$$C(F_1, F_2, S) = G_- \omega_- + G_+ \omega_+$$

Kinematical structure completely fixed, most algebraic simplifications (e.g. tensor reduction) can be carried out.

Classes level, e.g. `-F[2], F[1], S[3]`

$$\bar\ell_i \nu_j G: \quad G_- = -\frac{\mathrm{i}\, e\, m_{\ell,i}}{\sqrt{2}\sin\theta_w M_W}\delta_{ij}\,, \quad G_+ = 0$$

Coupling fixed except for i, j (can be summed in do-loop).

Particles level, e.g. `-F[2,{1}], F[1,{1}], S[3]`

insert fermion generation (1, 2, 3) for i and j

The Model Files

One has to set up, once and for all, a

- **Generic Model File** (seldomly changed) containing the generic part of the couplings,

Example: the `FFS` coupling

$$C(F, F, S) = G_- \omega_- + G_+ \omega_+ = \vec{G} \cdot \begin{pmatrix} \omega_- \\ \omega_+ \end{pmatrix}$$

```
AnalyticalCoupling[s1 F[j1, p1], s2 F[j2, p2], s3 S[j3, p3]]
== G[1][s1 F[j1], s2 F[j2], s3 S[j3]] .
     { NonCommutative[ ChiralityProjector[-1] ],
       NonCommutative[ ChiralityProjector[+1] ] }
```

The Model Files

One has to set up, once and for all, a

- **Classes Model File** (for each model)
  declaring the particles and the allowed couplings

Example: the $\bar\ell_i \nu_j G$ coupling in the Standard Model

$$\vec{G}(\bar\ell_i, \nu_j, G) = \begin{pmatrix} G_- \\ G_+ \end{pmatrix} = \begin{pmatrix} -\dfrac{\mathrm{i}\, e\, m_{\ell,i}}{\sqrt{2}\sin\theta_w M_W}\delta_{ij} \\ 0 \end{pmatrix}$$

```
C[ -F[2,{i}], F[1,{j}], S[3] ]
== { {-I EL Mass[F[2,{i}]]/(Sqrt[2] SW MW) IndexDelta[i, j]},
     {0} }
```

Current Status of Model Files

Model Files presently available for FeynArts:

- SM [w/QCD], normal and background-field version.
  All one-loop counter terms included.
- MSSM [w/QCD].
  Counter terms by T. Fritzsche.
- Two-Higgs-Doublet Model.
  Counter terms not included yet.
- ModelMaker utility generates Model Files from the Lagrangian.
- FeynRules package generates Model Files for FeynArts and other packages.

Partial (Add-On) Model Files

FeynArts distinguishes

- **Basic Model Files** and
- **Partial (Add-On) Model Files.**

Basic Model Files, e.g. `SM.mod`, `MSSM.mod`, can be modified by Add-On Model Files, for example,

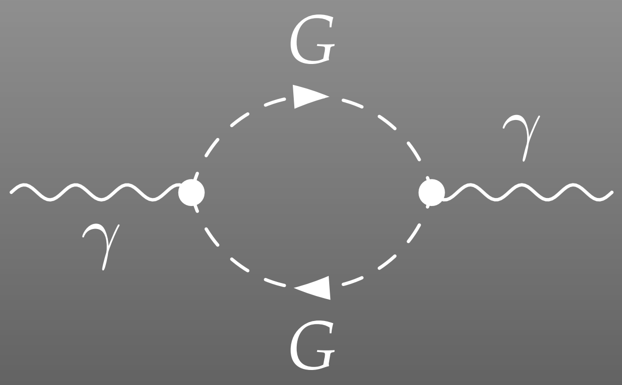
```
InsertFields[..., Model -> {"MSSMQCD", "FV", "HMix"}]
```

This loads the Basic Model File `MSSMQCD.mod` and modifies it through the Add-Ons `FV.mod` (non-minimal flavour violation) and `HMix.mod` (3×3 neutral Higgs mixing).

Model files can thus be built up from several parts.

- `M$ClassesDescription` = list of particle definitions,
- `M$CouplingMatrices` = list of couplings.

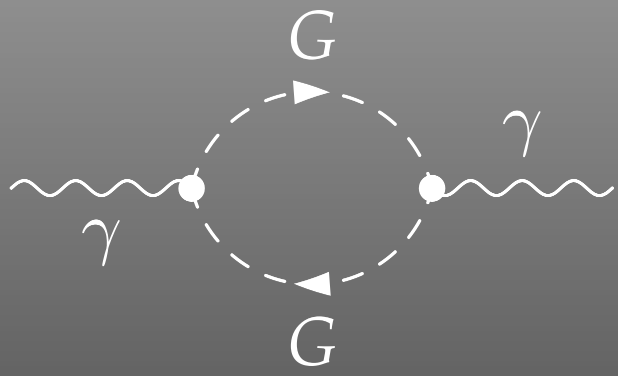
CreateFeynAmp output



= FeynAmp[*identifier* ,
loop momenta ,
generic amplitude ,
insertions]

```
GraphID[Topology == 1, Generic == 1]
```

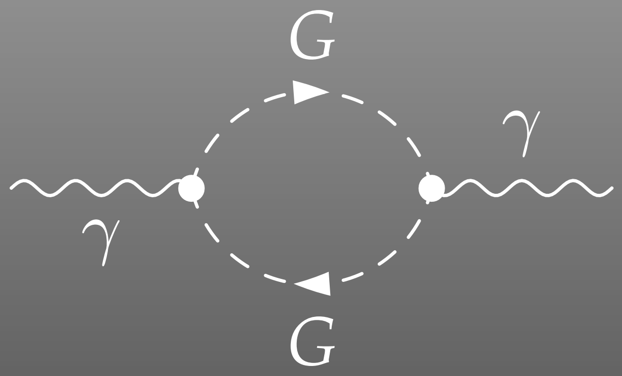
CreateFeynAmp output



= FeynAmp[*identifier*,
loop momenta,
generic amplitude,
insertions]

`Integral[q1]`

CreateFeynAmp output



= FeynAmp[*identifier*,
loop momenta,
generic amplitude,
insertions]

$\frac{\text{I}}{32\ \text{Pi}^4}$ RelativeCF .... *prefactor*

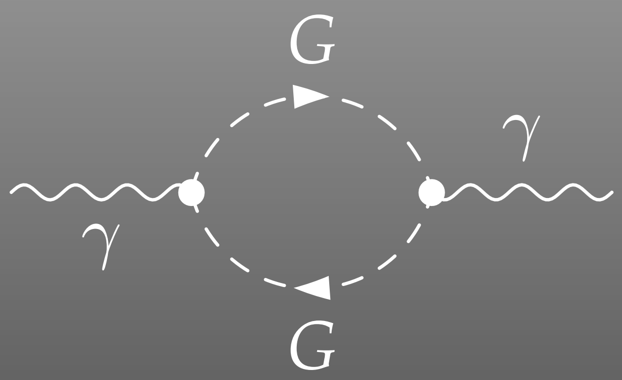
FeynAmpDenominator[$\frac{1}{\text{q1}^2 - \text{Mass[S[Gen3]]}^2}$,
$\frac{1}{(-\text{p1} + \text{q1})^2 - \text{Mass[S[Gen4]]}^2}$] .... *loop denominators*

(p1 - 2 q1)[Lor1] (-p1 + 2 q1)[Lor2] .... *kin. coupling structure*

ep[V[1], p1, Lor1] ep*[V[1], k1, Lor2] .... *polarization vectors*

$\text{G}^{(0)}_{\text{SSV}}$[(Mom[1] - Mom[2])[KI1[3]]]
$\text{G}^{(0)}_{\text{SSV}}$[(Mom[1] - Mom[2])[KI1[3]]] .... *coupling constants*

CreateFeynAmp output



= FeynAmp[*identifier*,
loop momenta,
generic amplitude,
insertions]

```
{ Mass[S[Gen3]],
  Mass[S[Gen4]],
  G(0)SSV[(Mom[1] - Mom[2])[KI1[3]]],
  G(0)SSV[(Mom[1] - Mom[2])[KI1[3]]],
  RelativeCF } ->
Insertions[Classes][{MW, MW, I EL, -I EL, 2}]
```

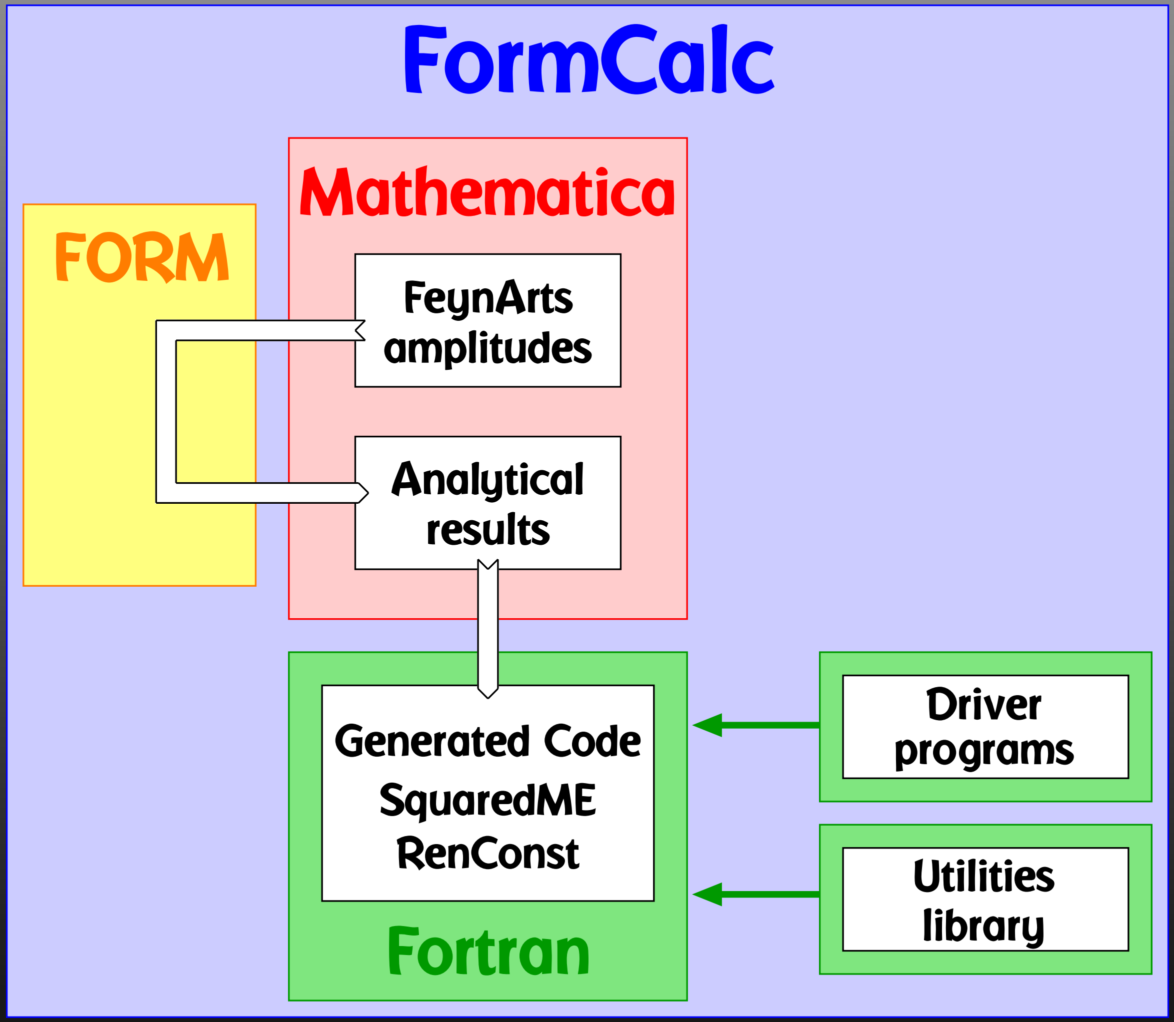
Algebraic Simplification

The amplitudes of `CreateFeynAmp` are in no good shape for direct numerical evaluation.

A number of steps have to be done analytically:

- contract indices as far as possible,
- evaluate fermion traces,
- perform the tensor reduction,
- add local terms arising from D·(divergent integral) (dim reg + dim red),
- simplify open fermion chains,
- simplify and compute the square of SU(N) structures,
- "compactify" the results as much as possible.

FormCalc Internals



FormCalc Output

A typical term in the output looks like

```
C0i[cc12, MW2, MW2, S, MW2, MZ2, MW2] *
  ( -4 Alfa2 MW2 CW2/SW2 S AbbSum16 +
    32 Alfa2 CW2/SW2 S^2 AbbSum28 +
    4 Alfa2 CW2/SW2 S^2 AbbSum30 -
    8 Alfa2 CW2/SW2 S^2 AbbSum7 +
    Alfa2 CW2/SW2 S (T-U) Abb1 +
    8 Alfa2 CW2/SW2 S (T-U) AbbSum29 )
```

[green] = loop integral	[red] = kinematical variables
[magenta] = constants	[yellow] = automatically introduced abbreviations

Abbreviations

Outright factorization is usually out of question.
Abbreviations are necessary to reduce size of expressions.

```
AbbSum29 = Abb2 + Abb22 + Abb23 + Abb3

Abb22 = Pair1 Pair3 Pair6

Pair3 = Pair[e[3],k[1]]
```

The full expression corresponding to `AbbSum29` **is**

```
Pair[e[1],e[2]] Pair[e[3],k[1]] Pair[e[4],k[1]] +
Pair[e[1],e[2]] Pair[e[3],k[2]] Pair[e[4],k[1]] +
Pair[e[1],e[2]] Pair[e[3],k[1]] Pair[e[4],k[2]] +
Pair[e[1],e[2]] Pair[e[3],k[2]] Pair[e[4],k[2]]
```

Categories of Abbreviations

- Abbreviations are recursively defined in several levels.
- When generating Fortran code, FormCalc introduces another set of abbreviations for the loop integrals.

In general, the abbreviations are thus costly in CPU time.
It is key to a decent performance that the abbreviations are separated into different Categories:

- Abbreviations that depend on the helicities,
- Abbreviations that depend on angular variables,
- Abbreviations that depend only on $\sqrt{s}$.

Correct execution of the categories guarantees that almost no redundant evaluations are made and makes the generated code essentially as fast as hand-tuned code.

The Abbreviate Function

The `Abbreviate` **Function** allows to introduce abbreviations for arbitrary (sub-)expressions and extends the advantage of categorized evaluation. Example:

```
abbrexpr = Abbreviate[expr, 5]
```

The second argument, 5, determines the **Level** below which abbreviations are introduced, i.e. how much of expression is 'abbreviated away.' Abbreviationing has the 'side effect' that **duplicate expressions are replaced by the same symbol.**

The subexpressions are **retrieved with `Subexpr[]`.**

Typical speed-ups are a **factor 3** in MSSM calculations at typical execution times of `Abbreviate` of 30 sec.

Re-using Abbreviations

Abbreviations were so far restricted to one FormCalc session, e.g. one could not save intermediate results involving abbreviations and resume computation in a new session.

FormCalc 6 adds two functions to 'register' abbreviations and subexpressions from an earlier session:

```
RegisterAbbr[abbr]
RegisterSubexpr[subexpr]
```

Alternate Link between FORM and Mathematica

FORM is renowned for being able to handle very large expressions. To produce (pre-)simplified expressions, however, terms have to be wrapped in functions, to avoid immediate expansion. The number of terms in a function is severely limited in FORM: on 32-bit systems to 32767.

Dilemma: FormCalc gets more sophisticated in pre-simplifying amplitudes while users want to compute larger amplitudes. Thus, users have recently seen many 'overflow' messages from FORM.

Solution: Pre-simplified generic amplitude is sent to Mathematica intermediately for introducing abbreviations. Result: significant reduction in size of intermediate expressions.

Tentukov, Vermaseren 2006

Effect on Intermediate Amplitudes

FORM → Mathematica:

part of $uu \to gg$ @ tree level

```
+Den[U,MU2]*(
  -8*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*Pi]*
  abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,Spinor[k2,MU,1]]]
    +fme[WeylChain[DottedSpinor[k1,MU,-1],7,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,Spinor[k2,MU,1]]]]*MU
  -4*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*Pi]*
  abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,k3,Spinor[k2,MU,1]]]
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec4,Spinor[k2,MU,1]]]*ec3.k2
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,k3,Spinor[k2,MU,1]]]*ec3.ec4
    +fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,k3,Spinor[k2,MU,1]]]
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec4,Spinor[k2,MU,1]]]*ec3.k2
    -2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,k3,Spinor[k2,MU,1]]]*ec3.ec4]
  +8*SUNSum[Col5,3]*SUNT[Glu3,Col5,Col2]*SUNT[Glu4,Col1,Col5]*mul[Alfas*MU*Pi]*
  abb[fme[WeylChain[DottedSpinor[k1,MU,-1],6,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],6,ec3,ec4,Spinor[k2,MU,1]]]
    +fme[WeylChain[DottedSpinor[k1,MU,-1],7,Spinor[k2,MU,1]]]*ec3.ec4
    -1/2*fme[WeylChain[DottedSpinor[k1,MU,-1],7,ec3,ec4,Spinor[k2,MU,1]]]] )
```

Mathematica → FORM:

```
-4*Den(U,MU2)*SUNSum(Col5,3)*SUNT(Glu3,Col5,Col2)*SUNT(Glu4,Col1,Col5)*
  AbbSum5*Alfas*Pi
```

CutTools

Tensor loop integrals have in FormCalc so far been treated by **Passarino-Veltman reduction** only, e.g.

$$\frac{q_\mu q_\nu}{D_0 D_1} = g_{\mu\nu}\,\mathtt{B00}(p, m_1, m_2) + p_\mu p_\nu\,\mathtt{B11}(p, m_1, m_2)$$

where `B00` and `B11` are **provided by LoopTools.**

CutTools implements the cutting-technique-inspired OPP (Ossola, Papadopoulos, Pittau) method. It needs the numerator as a function of q which it can sample:

$$\mathtt{Bcut}(2, \mathtt{num1}, \mathtt{num2}, p, m_1, m_2)$$

where $\mathtt{num1} = q_\mu q_\nu$ and $\mathtt{num2} = 0$ (coeff. of $D - 4$).

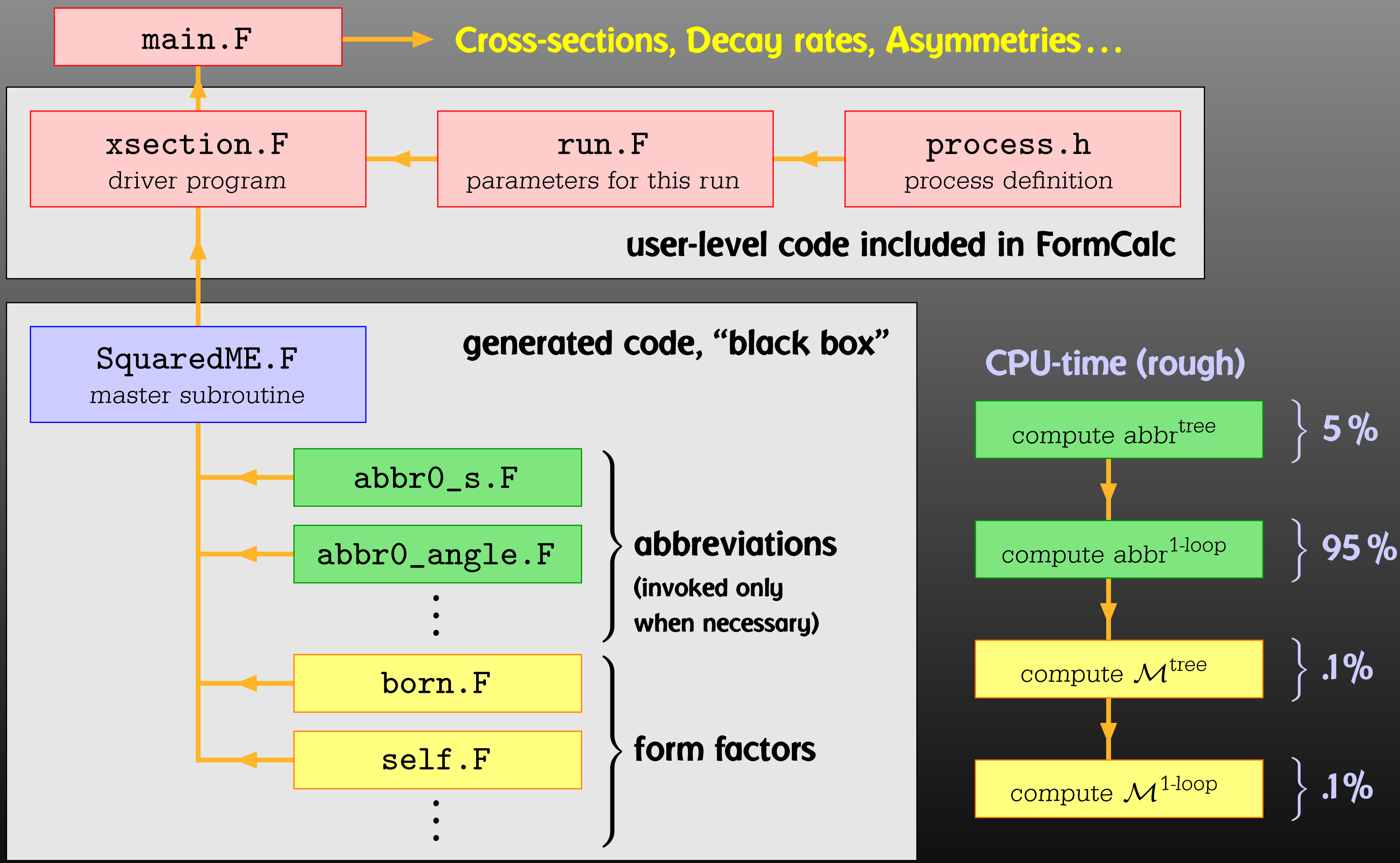
Independent way of checking LoopTools results.
Performance?

Dirac Chains in 4D

As numerical calculations are done mostly using Weyl-spinor chains, there has been a paradigm shift for Dirac chains to make them better suited for analytical purposes, e.g. the extraction of Wilson coefficients.

- Already in Version 5, Fierz methods have been implemented for Dirac chains, thus allowing the user to force the fermion chains into almost any desired order.
- Version 6 further adds the `Colour` method to the `FermionOrder` option of `CalcFeynAmp`, which brings the spinors into the same order as the external colour indices.
- Also new in Version 6: completely antisymmetrized Dirac chains, i.e. $\texttt{DiracChain}[-1, \mu, \nu] = \sigma_{\mu\nu}$.

Numerical Evaluation in Fortran 77



Features of the Generated Code

- **Modular:** largely autonomous pieces of code provide
  - kinematics,
  - model initialization,
  - convolution with PDFs.
- **Extensible:** default code serves (only) as an example. Other ‘Frontends’ can be supplied, e.g. HadCalc, sofox.
- **Re-usable:** external program need only call `ProcessIni` (to set up the process) and `ParameterScan` (to set off the calculation).
- **Interactive:** Mathematica interface provides Mathematica function for cross-section/decay rate.
- **Parallel:** built-in distribution of parameter scans.

Summary

New Features in FormCalc Version 6:

- Intermediate FORM expressions get sent to Mathematica for abbreviations – significant size reduction.
- New Functions for registering abbreviations and subexpressions allow to 'resume' sessions.
- Improvements in 4D Dirac chains for analytical purposes (choice of ordering, antisymmetrization).
- Significant improvements in code-generation routines. They are independent of other features and turn out production-quality code for other applications.
- Version 6 is publicly available: http://www.feynarts.de/formcalc

CutTools To-do List

- **Finished: Code generation** for linking with the CutTools library. Highly optimized w.r.t. sampling of numerator, e.g.

```
      subroutine Num15(res,q1in)
      implicit none
      double complex res, q1in(0:3)

#include "num.h"

      res = 1/8.D0*QC18 - 1/8.D0*(QC17*Eps(q1,e(1),ec(2),k(1))) -
    -   1/4.D0*(QC15*Pair(q1,e(1))*Pair(q1,ec(2))) -
    -   1/8.D0*(QC16*Pair(q1,k(1)))
      end
```

- **To do: Adapt library conventions** (naming, Minkowski vs. light-cone vectors, etc.)

- **To do: Add dimensionally regularized IR divergences** to LoopTools (60% done).

  Technically: LTLAMBDA $= -2, -1, 0$ returns dim. reg. IR poles,
  LTLAMBDA > 0 photon-mass regularization.